Philippine KLING

PROJET 5 : CRÉATION DE TAGS POUR LES QUESTIONS DE STACK OVERFLOW

Sujet : Stack Overflow est un site célèbre de questions-réponses liées au développement informatique. Pour poser une question sur ce site, il faut entrer plusieurs tags de manière à retrouver facilement la question par la suite. Pour les utilisateurs expérimentés, cela ne pose pas de problème, mais pour les nouveaux utilisateurs, il serait judicieux de suggérer quelques tags relatifs à la question posée.

Amateur de Stack Overflow, qui vous a souvent sauvé la mise, vous décidez d'aider la communauté en retour. Pour cela, vous développez un système de suggestion de tag pour le site. Celui-ci prendra la forme d'un algorithme de machine learning qui assigne automatiquement plusieurs tags pertinents à une question.

PARTIE 1 : récupérer les données

Afin de proposer quelques tags à un nouveau post sur le site internet, nous allons commencer par analyser ceux qui l'ont déjà été. Cela nous permettra de comprendre un peu mieux la demande.

Pour cela nous allons récupérer quelques posts ainsi que les tags qui leur sont associés via la plateforme de récupération des données de stack overflow. Nous choisissons de baser notre étude sur des post qui ont attiré l'attention des utilisateurs, soit qui ont "réussi". Pour cela nous ne prendrons que ceux dont le score est supérieur à 0, qui ont un utilisateur, un titre et un corp de texte non null.

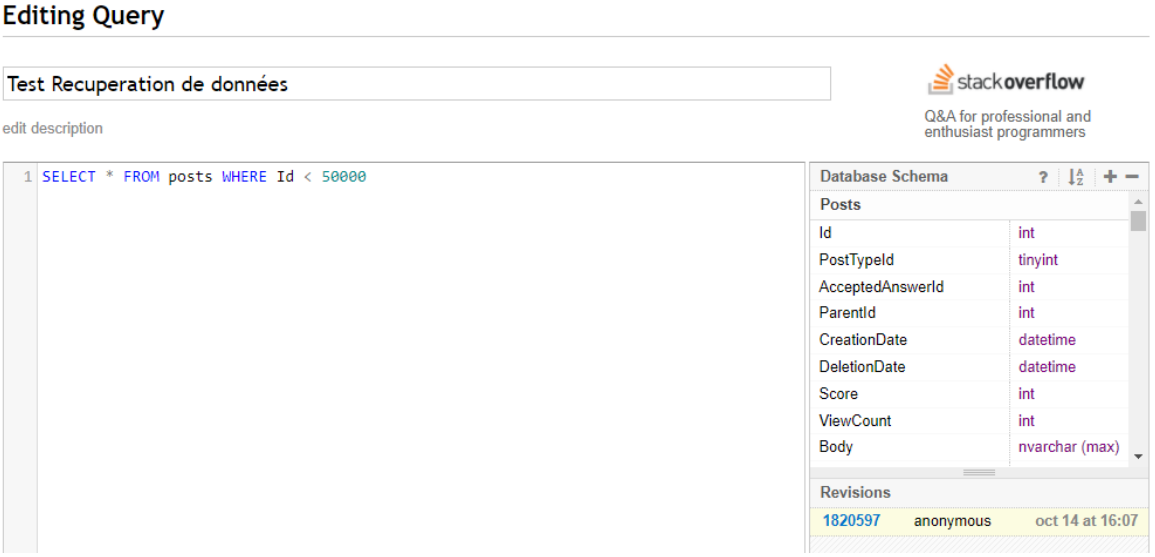
Récupération des questions :

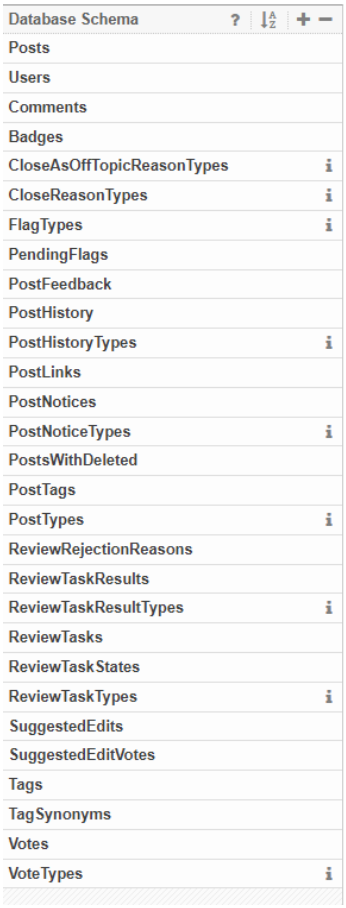
```
SELECT Id, OwnerUserId, CreationDate, ClosedDate, Score, Title, Body FROM posts
WHERE Id < 50000
AND OwnerUserId is not Null
AND Score > 0
AND Title is not Null
```

Récupération des Tags

```
SELECT PostId, TagName
FROM  PostTags left join Tags on TagId = Tags.Id
WHERE PostId < 50000
```

Nous récupérons les questions et les tags dans deux csv différents de façon à alléger la requête





La bibliothèque SQL proposée par StakeoverFlow propose un large choix de tables. Comme ce sont les données Post et les données Tags qui nous intéressent et que la taille de l'importation est limitée nous avons choisi de nous limiter à une simple joiture entre ces deux tables.

Cependant il pourra être intéressant de s'intéresser au utilisateurs, voir si le score de leur post augmente au fil des posts et si la qualité des tags employés augmente aussi

PARTIE 2 : EDA / Analyse exploratoire

Dans cette partie nous allons faire une petite analyse préparatoire des différents post et tasg récupérés

Analyse exploratoire du fichier des posts :

Le fichier comporte 7 colonnes, comme le montre la capture des premières lignes ci-dessous :

Id	OwnerUserId	CreationDate	ClosedDate	Score	Title	Body
4	8	2008-07-31 21:42:52	NaN	752	How to convert a Decimal to a Double in C#?	<p>I want to use a <code>Track-Bar</code> to c...
6	9	2008-07-31 22:08:08	NaN	312	Why did the width collapse in the percentage w...	<p>I have an absolutely positioned <code>div</...
9	1	2008-07-31 23:40:59	NaN	2083	How do I calculate someone's age based on a Da...	<p>Given a <code>DateTime</code> representing ...
11	1	2008-07-31 23:55:37	NaN	1599	Calculate relative time in C#	<p>Given a specific <code>DateTime</code> valu...
13	9	2008-08-01 00:42:38	NaN	667	Determine a user's timezone	<p>Is there a standard way for a web server to...

Observations :
Un score permet de savoir si la question a été utile à d'autres internautes
Le Titre est une simple string
Le Body par contre comporte des balises HTML

	Score
count	5031.000000
mean	70.814550
std	315.624477
min	1.000000
25%	4.500000
50%	11.000000
75%	33.000000
max	7318.000000

Le score est compris entre 1 et 33, avec quelques valeurs exceptionnelle qui peuvent le faire dépasser le millier
Il peut être intéressant de regarder le nombre de tags choisi par les utilisateurs "experts", c'est-à-dire ceux dont les scores ont dépassé le millier.

```
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Id            5031 non-null   int64
 1   OwnerUserId   5031 non-null   int64
 2   CreationDate  5031 non-null   object
 3   ClosedDate    704 non-null    object
 4   Score         5031 non-null   int64
 5   Title         5031 non-null   object
 6   Body          5031 non-null   object
dtypes: int64(3), object(4)
memory usage: 275.3+ KB
```

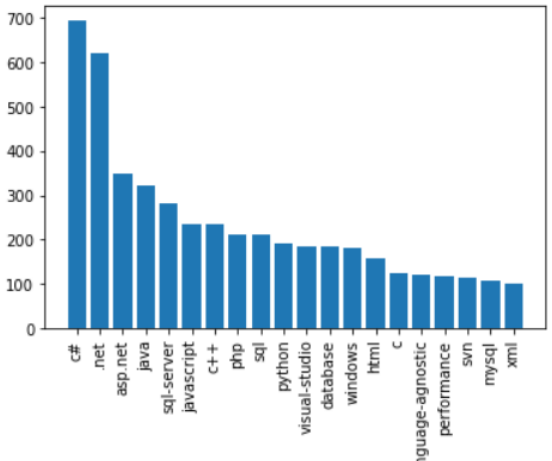
On vérifie avec la fonction df.info() que le dataframe ne comporte pas de valeur null, ce qui semble bien être le cas

Fichier Tags :

Nous regroupons tous les tags associés à une même question pour avoir un aperçu du rendu. Par exemple, le post 9 comprend 3 tags : "datetime" "C#" et ".net"

```
Id
4     c# decimal double floating-point type-conversion
6                          html internet-explorer-7 css
9                                      datetime c# .net
11        c# datetime datediff time relative-time-span
13    timezone timezone-offset user-agent browser html
14                                          math .net
16                       .net-3.5 c# linq web-services
17              mysql binary-data data-storage database
19     algorithm performance pi language-agnostic unix
24                           triggers mysql database
```

On cherche ensuite à regarder quels sont les tags les plus courants: étonnement dans l'exportation utilisée, c'est le tag "C#" qui est le plus courant, et non le tag "PYTHON"



Le fichier Tags comporte deux colonnes, Id du post auquel il est relié, et Nom du tag. On remarque que sur les 2,6 mille différents tags, seul quelques centaines sont vraiment fréquents et permettent de couvrir la plupart des sujets.

Most frequent tags

```
number of Tags : 2641
new number of Tags : 239
```

Nous choisissons donc de ne garder que les tags suffisamment fréquent ce qui risque de faire baisser le nombre de post pour le traitement prochain mais est essentiel pour gagner du temps de calcul et garder des tags cohérents et non exceptionnels. Et diminuer le nombre de class

PARTIE 3 : Prétraitement des données

Nous avons besoin pour prédire les données de deux colonnes : Une colonne avec la question et une colonne avec la liste des tags pour calculer des scores pour les méthodes supervisées

Il faut ensuite laver les textes de sorte à les simplifier, voici la méthode suivie, étudiée en cours

```
def nettoyage_text(text):
    text = retire_html(text)
    text = retire_punck(text)
    text = lemitize_words(text)
    text = retire_stopwords(text)
    return text
```

Pour cela on procède par étape :

- retirer les balises HTML du texte
- retirer les majuscules et la ponctuation du texte. Sans retirer les # des C#
- lemmatisation du texte (récupération des radicaux d'un mot)
- retirer les mots qui ne sont pas porteur de sens

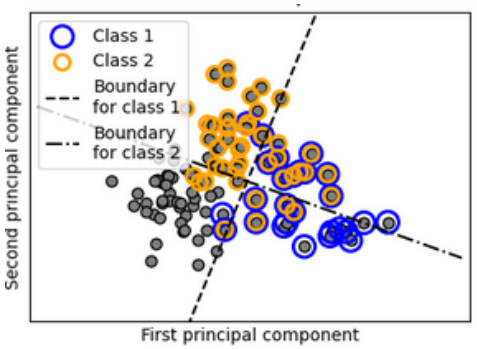
PARTIE 4 : entrainer les modèles qui vont prédire les tags

Modèle intuitif sans classification

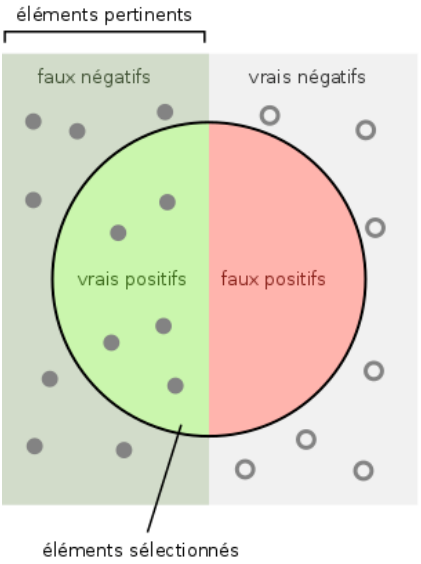
Ce modèle consiste, une fois le text traité, à récupérer les mots/lemme les plus courants

classification classique et supervisée

OnevsRestClassifier : Classifier qui utilise un classifier par class. Ce qui permet à un élément d'être relié à plusieurs class et dans notre cas est très intéressant pour proposer plusieurs tags à l'utilisateur



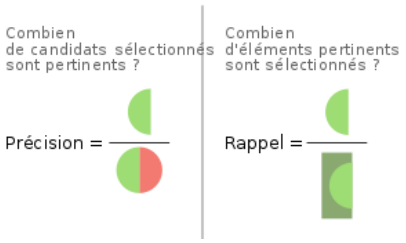
Deux autres classifier plus classiques ont été utilisé à titre de comparaison :
RandomForest : classifier basés sur des arbres de décision randomisés
MLP Classifier : Réseau de neurone basé sur un apprentissage itératif



<u>Les scores utilisés :</u>

$$J(y_i, \hat{y}_i) = \frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|}$$

Score de Jacard :
score de Hamming : Le score de Hamming est la fraction des prédictions correctes par rapport au total des étiquettes
Accuracy : Dans la classification multi-label, cette fonction calcule la précision du **sous-ensemble** : l'ensemble des étiquettes prédites pour un échantillon doit correspondre exactement à l'ensemble correspondant des étiquettes dans y_true.
Recall (ou Rappel): il permet de savoir le pourcentage de positifs bien prédit par notre modèle
Précision : nbr d'élément positif bien prédit parmis tous les éléments prédit



F1 score :

$$F1\ Score = 2 \times \frac{recall \times precision}{recall + precision}$$

	Classifier	Jacard_avg	Jacard_macro	Hamming	Accuracy	Recall	precision	Time	f1
0	OneVsRestClassifier	0.060561	0.032848	0.123412	0.017725	0.063432	0.061175	0 days 00:00:00.039032	0.096268
1	OneVsRestClassifier	0.482029	0.429095	0.047046	0.370753	0.491359	0.722980	0 days 00:00:00.090711	0.582295
2	OneVsRestClassifier	0.306992	0.265045	0.049483	0.249631	0.271836	0.818457	0 days 00:00:00.345710	0.440735
3	OneVsRestClassifier	0.077548	0.058747	0.062555	0.064993	0.059452	0.495069	0 days 00:00:00.036861	0.125903
4	OneVsRestClassifier	0.469719	0.418337	0.044904	0.367799	0.460648	0.786326	0 days 00:00:00.127232	0.582418
5	OneVsRestClassifier	0.435106	0.384485	0.064919	0.264402	0.540927	0.538948	0 days 00:00:00.090086	0.520458
6	OneVsRestClassifier	0.448227	0.396025	0.061891	0.292467	0.525118	0.572987	0 days 00:00:00.308482	0.526554
7	MLPClassifier	0.449902	0.400592	0.051551	0.327917	0.464944	0.710217	0 days 00:00:13.230721	0.548512
8	RandomForestClassifier	0.400788	0.313318	0.045790	0.339734	0.323243	0.801339	0 days 00:00:03.595457	0.507937

Liste des classifier utilisés par le OnevsRest :

0 - dummy = DummyClassifier()
1 - sgd = SGDClassifier()
2 - lr = LogisticRegression()
3 - mn = MultinomialNB()
4 - svc = LinearSVC()
5 - perceptron = Perceptron()
6 - pac = PassiveAggressiveClassifier()

Après optimisation du meilleur classifieur OneVsRestClassifer et utilisation du LinearSVC() via une Gridsearch
Le classifier est optimisé pour un hyperparamètre C=1, et un score de Jacard de 49.7 et 0.1s

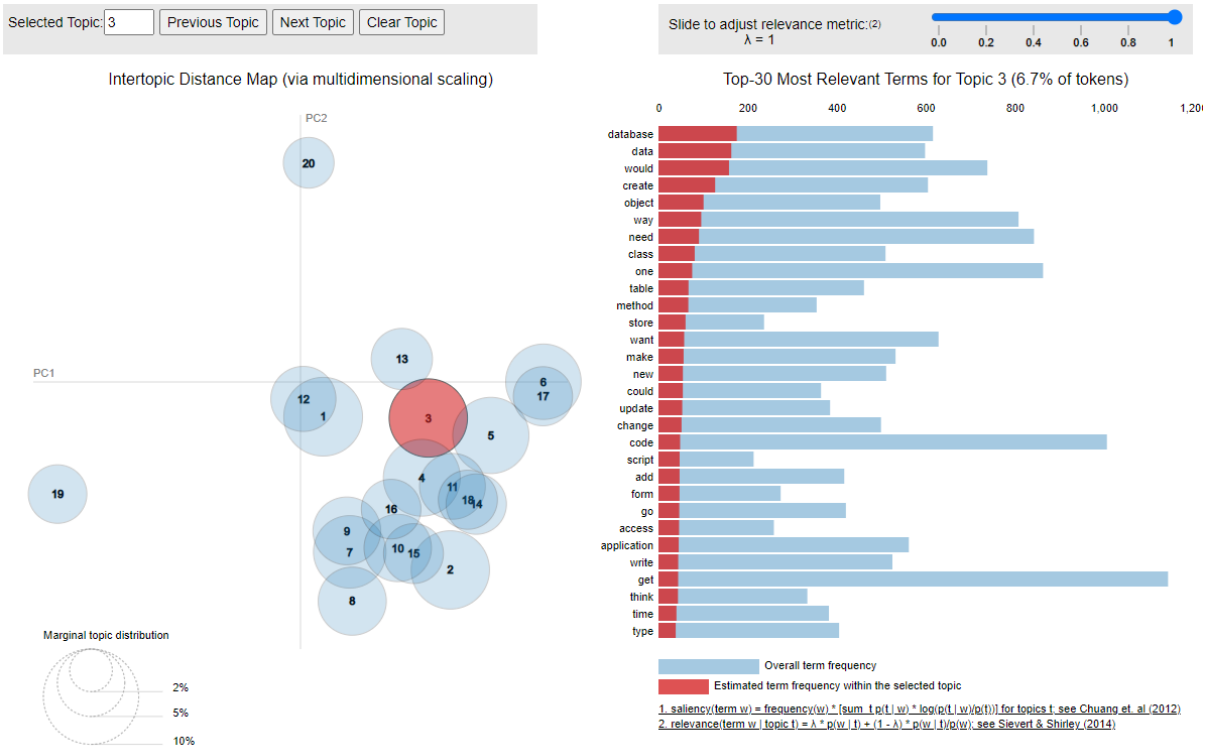
Voici les catégories récupérée :

```
.net: represent syntax domain vista native via generic wcf assembly net
asp.net: side section postback control page cache cause web webconfig aspnet
c: level argument include b i gcc char platform c cc
c#: thoughts listview namespace way compact foreach lock core datatable c
c++: linux certain exist bool 64 include const win32 c cc
database: join char trick transaction language record past databases db database
html: tag batch font site give inline put width div html
java: api distribute application happen look action dependency javadoc eclipse java
javascript: function ajax js virtual side browser var ie jquery javascript
language-agnostic: statement regex similar case non prefer function ways based language
mysql: integration in live entry support 3 statements limit query mysql
performance: site efficient scale sense time fastest faster optimize speed performance
php: live specific post integration resize entry apache permissions session php
python: thing plug x none iterate drag parameters equivalent django python
sql: length based time graph choice index select insert join sql
sql-server: insert procedure script ms 2005 master fill temp server sql
svn: source proxy development control merge version branch repository subversion svn
visual-studio: custom textbox ui flag project team 2003 debug studio visual
windows: question user export standard core task thoughts drive software windows
xml: find replace section choice quite comment apply node nod xml
```

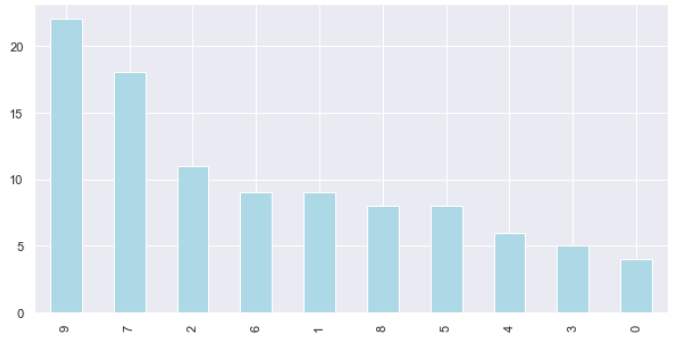
Classification non supervisée LDA

le LDA. Algorithme non supervisé qui permet de classifier en sujet les différents texte et donne les mots les plus fréquents de chaque sujet

Projection des données d'entrée dans un sous-espace linéaire constitué des directions qui maximisent la séparation entre les classes (dans un sens précis discuté dans la section mathématique ci-dessous). La dimension de la sortie est nécessairement inférieure au nombre de classes, il s'agit donc d'une réduction de dimensionnalité assez forte, qui n'a de sens que dans un cadre multi-classes.



Sur un extrait de 100 questions, on voit que tous les sujets ne ressortent pas à la même fréquence

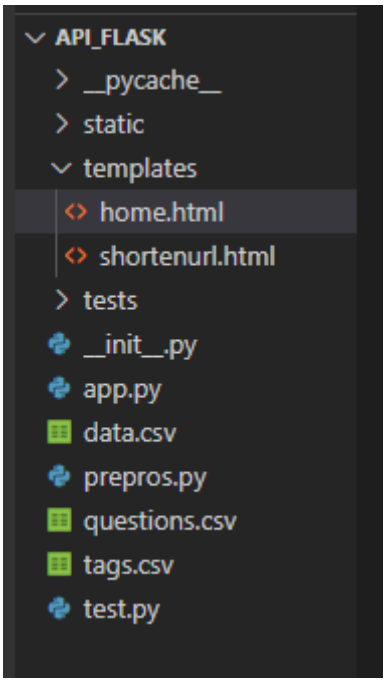


PARTIE 4 : présenter les résultats sous forme d'une API

- API : développer un modèle dvp sur un serveur, si on envoie le modèle à tel adresse api "resfull api" de flask : https://flask-restful.readthedocs.io/en/latest/ fast api

Pour développer l'API je m'y suis prise en plusieurs étapes :

- création d'un répertoire avec les dossiers principaux
- écriture en HTML des pages d'entrée des données (home.html) et de récupération des tags (shortenurl.html)
- création des fichier .py de pré-processing (prepros.py) et de prédiction de tags (test.py)
- code du ficher app.py de sorte à lancer le site



Pour le dépôt des informations : titre et corp de la question, j'ai choisi de créer un questionnaire avec deux cases et un bouton "submit" afin de lancer la recherche de tags

home.html shortenurl.html

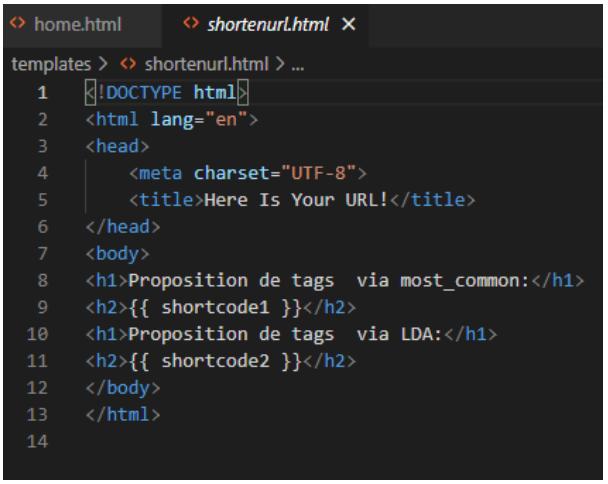
templates > home.html > html > body > form

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>API Stack Overflow</title>
</head>
<body>
<h1>Prédiction de tags Stack Overflow</h1>
<form action="shortenurl" method="post">
    <!-- Champ pour le titre -->
    <label for="title">Title</label>
    <input type="text" name="title" value="" required>
    <!-- Champ pour le développement -->
    <label for="question">Question</label>
    <input type="text" name="question" size="100" value="" required>
    <!-- Champ pour soumettre -->
    <input type="submit" value="Submit">
</form>

</body>
</html>
```

Pour la réponse j'ai décidé de mettre les tags proposés par les deux méthodes :

- recherche des mots les plus utilisés
- classification non supervisée



Cela permet de comparer les résultats obtenus par les deux méthodes.

Conclusion :

Les avantages

- Ce projet m’a permis d’utiliser les bibliothèques de traitement de texte
- La LDA permet de proposer des tags qui ne sont pas forcément dans le texte.

Les pistes d’améliorations

- Il faudrait utiliser plus de données pour la classification LDA de sorte à créer plus de catégories différentes. Par contre cela risque d’augmenter significativement le temps de calcul du modèle
- travailler sur les synonymes des tags
- utiliser les réponses comme élément de la question pour les modèles d'entraînement